AD-A076 418 YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE F/G 5/7
PROBLEMS IN CONCEPTUAL ANALYSIS OF NATURAL LANGUAGE.(U)
OCT 79 L BIRNBAUM , M SELFRIDGE N00014-75-C-1111
UNCLASSIFIED RR-168 NL

1 OF 1
AD-
A076418

END
DATE
FILMED
12-79
DDC

12

PROBLEMS IN CONCEPTUAL ANALYSIS
OF NATURAL LANGUAGE

October 1979

Research Report #168

Lawrence Birnbaum and Mallory Selfridge

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

PROBLEMS IN CONCEPTUAL ANALYSIS OF NATURAL LANGUAGE

October 1979

Research Report #168

Lawrence Birnbaum and Mallory Selfridge

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and Monitored under the Office of Naval Research under contract N00014-75-C-1111.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(9) Research rept.,

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER #168	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Problems in Conceptual Analysis of Natural Language		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Lawrence Birnbaum and Mallory Selfridge		8. CONTRACT OR GRANT NUMBER(s) (15) N00014-75-C-1111
9. PERFORMING ORGANIZATION NAME AND ADDRESS Yale University - Department of Computer Science 10 Hillhouse Avenue New Haven, Connecticut 06520		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 68
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE (11) October 1979
		13. NUMBER OF PAGES 63
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Program Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

(14) RR-168

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Natural Language Analysis
Parsing
Conceptual Analysis
Artificial Intelligence

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper reports on some recent developments in natural language analysis. We address such issues as the role of syntax in a semantics-oriented analyzer, achieving a flexible balance of top-down and bottom-up processing, and the role of short term memory. Our results have led to improved algorithms capable of analyzing the kinds of multi-clause inputs found in most text.

407 051 JOB

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

-- OFFICIAL DISTIRUBTION LIST --

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 copies
Judith Daly Advanced Research Projects Agency Cybernetics Technology Office 1400 Wilson Boulevard Arlington, Virginia 22209	3 copies
Office of Naval Research Branch Office - Boston 495 Summer Street Boston, Massachusetts 02210	1 copy
Office of Naval Research Branch Office - Chicago 536 South Clark Street Chicago, Illinois 60615	1 copy
Office of Naval Research Branch Office - Pasadena 1030 East Green Street Pasadena, California 91106	1 copy
Mr. Steven Wong Administrative Contracting Officer New York Area Office 715 Broadway - 5th Floor New York, New York 10003	1 copy
Naval Research Laboratory Technical Information Division Code 2627 Washington, D.C. 20375	6 copies
Dr. A.L. Slafkosky Scientific Advisor Commandant of the Marine Corps Code RD-1 Washington, D.C. 20380	1 copy
Office of Naval Research Code 455 Arlington, Virginia 22217	1 copy

Office of Naval Research Code 458 Arlington, Virginia 22217	1 copy
Naval Electronics Laboratory Center Advanced Software Technology Division Code 5200 San Diego, California 92152	1 copy
Mr. E.H. Gleissner Naval Ship Research and Development Computation and Mathematics Department Bethesda, Maryland 20084	1 copy
Captain Grace M. Hopper NAICOM/MIS Planning Board Office of the Chief of Naval Operations Washington, D.C. 20350	1 copy
Advanced Research Project Agency Information Processing Techniques 1400 Wilson Boulevard Arlington, Virginia 22209	1 copy
Professor Omar Wing Columbia University in the City of New York Department of Electrical Engineering and Computer Science New York, New York 10027	1 copy
Office of Naval Research Assistant Chief for Technology Code 200 Arlington, Virginia 22217	1 copy
Captain Richard L. Martin, USN Commanding Officer USS Francis Marion (LPA-249) FPO New York 09501	1 copy
Major J.P. Pennell Headquarters, Marine Corp. (Attn: Code CCA-40) Washington, D.C. 20380	1 copy
Computer Systems Management, Inc. 1300 Wilson Boulevard, Suite 102 Arlington, Virginia 22209	5 copies

PROBLEMS IN CONCEPTUAL ANALYSIS OF NATURAL LANGUAGE

Lawrence Birnbaum and Mallory Selfridge

ABSTRACT

This paper reports on some recent developments in natural language analysis. We address such issues as the role of syntax in a semantics-oriented analyzer, achieving a flexible balance of top-down and bottom-up processing, and the role of short term memory. Our results have led to improved algorithms capable of analyzing the kinds of multi-clause inputs found in most text.

1.0 INTRODUCTION

This paper reports on some recent developments in conceptual analysis of natural language. Work on conceptual analysis has been proceeding for several years now, starting with the work reported in Schank et al. (1970) and Riesbeck (1975). We have built on that work, concentrating on some important theoretical issues that we feel have not received sufficient attention:

(1) What is the role of syntax in a semantics-oriented language analysis process?

(2) What is the proper relationship between top-down and bottom-up modes of operation?

(3) How does short-term memory structure affect the analysis process?

(4) What is the right vocabulary (formalism) for expressing the processing knowledge that the system has?

(5) What are the criteria on the performance of a word-sense disambiguation process which we should strive to achieve?

Our work has provided some answers to the above questions, which have in turn enabled us to construct more flexible algorithms, capable of analyzing the kinds of complex noun groups and multi-clause inputs found in most text. It was precisely this sort of input which caused problems in previous conceptual analysis programs.

1.1 Outline of the paper

2.0 BACKROUND

The goal of the conceptual analysis process is to map natural language input into a representation of its meaning (see Schank (1975)). We are thus concerned with explaining how a machine or human might derive the meaning of some input text based upon the meanings of the words and phrases from which it is composed, plus other knowledge. This goal differs from that assumed by most models of parsing, which hold that the purpose of the parsing mechanism is to perform a syntactic analysis of the input sentences. In other words, the goal of a conceptual analyzer is to derive the semantic and memory structures underlying an utterance, whereas the goal of a syntactic parser is to discover its (syntactic) structural description.[1] However, since any understanding system must eventually transform text into some internal semantic representation, it is clear that conceptual analysis subsumes the ultimate purpose of traditional syntactic parsing algorithms. We claim that such prior syntactic analysis is unnecessary, and that for this and other reasons it is not a plausible part of a model of human language understanding. This, of course, does not mean that we deny the existence of syntactic phenomena or of syntactic knowledge: we are arguing against the notion of a separate syntactic analysis phase.

----

(1) See, for example, Thorne et al. (1968), Bobrow and Fraser (1969), Woods (1970), Winograd (1972), Kaplan (1975), Marcus (1975), Marcus (1979). Our own point of view is derived from the work of Schank and Riesbeck. Previous work is described in Schank et al. (1970), Riesbeck (1975), Riesbeck and Schank (1976), Gershman (1977) and (1979). For somewhat different work with similar motivations, see Wilks (1973) and (1976). For related work with differing motivations, see Burton (1976).

Since the goal of conceptual analysis differs from the goal of syntactic parsing, it is understandable that the mechanisms which are employed differ as well. In particular, it is our goal that the conceptual analysis process should use any and all available knowledge whenever helpful. Such knowledge may include syntactic knowledge, but is certainly not limited to that. Because the goal of the process is no longer seen as performing syntactic analyses (which are typically supposed to serve as input to some semantic interpreter), there is no hint of circularity in the use of non-syntactic information. It is a claim of the theory that non-linguistic knowledge is often crucial even at the earliest points in natural language processing.

Because the goal of conceptual analysis is to map natural language into a conceptual representation of the meaning, an adequate representation must be specified. In what follows, we will employ conceptual dependency (CD) (see Schank (1975), Schank and Abelson (1977)). However, we believe that the principles and mechanisms of conceptual analysis are to a large extent independent of the particular representation chosen, as long as that representation meets certain criteria. Almost any well-developed system based on case frames (see Bruce (1975)) seems adequate. Case frame representations are useful because they naturally contain a great deal of information on the possible connections between any items so represented, in the form of unfilled case slots.

2.1 Expectations

How can we characterize, in a rough sense, the process of language understanding? A language understander must connect concepts which are obtained from word meanings, and from inferences derived from word meanings, into a coherent representation of the input as a whole. Because of the possibilities of word-sense ambiguity or irrelevent inference, an understander must also be able to choose from among alternative concepts. Thus, conceptual analysis consists mainly of connecting and disambiguating (i.e., choosing among) representational structures.

The processing knowledge which the system uses for this task are expectations (see Schank et al. (1970), Riesbeck (1975), Riesbeck and Schank (1976)). When a person hears or reads words with certain meanings, he expects or predicts that words with certain other meanings may follow, or may have already been seen. People constantly form expectations, predictions as to what they are likely to see next, based on what they have read and understood so far and on what they know about language and the world, and they use these expectations to disambiguate and connect incoming text.

Of course, expectations are used in syntactic analysis programs as well. The difference lies in the origin of the expectations. In syntactic analyzers, the expectations are derived from a grammar. In Schank et al. (1970), it was recognized that the expectations should be governed by the incomplete case structures representing the meaning of the input.

To illustrate the use of expectations in understanding, suppose that the following simple sentence were the input to an expectation-based conceptual analysis system: "Fred ate an apple." Reading from left to right, the system first finds the word "Fred". Since there are as yet no expectations, the system simply understands this as a reference to some male human being named Fred, and stores the reference, represented as the token FRED, in some kind of short term memory. The next word is "ate". This is understood as an instance of the concept of eating, which in conceptual dependency is represented by a case frame something like this: (INGEST ACTOR (NIL) OBJECT (NIL)). Also, the meaning of "ate" supplies some expectations which give hints as to how to go about filling the unfilled slots of this case frame. One of these expectations suggests that the ACTOR of the INGEST, an animate being, may have already been mentioned. So the analyzer checks short term memory, finds FRED there, and fills the ACTOR slot of the INGEST: (INGEST ACTOR (FRED) OBJECT (NIL)).

There remains an unfulfilled expectation, which suggests that some mention will be made of what it is that Fred ate, that it should be some edible thing (or at least a physical object), and that it should fill the OBJECT slot. Next, the word "an" is read. This creates an expectation that some instance of a concept should follow, and instructs that if one is found, it should be marked as a new instance of that concept. (This is information which can aid memory.) Finally, "apple" is read. It is understood as an instance of the concept APPLE, representing something which is known to be food. The expectation created when "an" was read is satisfied, so APPLE is marked as an object not previously seen, which we will represent as:

(APPLE REF (INDEF)). The second expectation created when "ate" was read is also satisfied, so the OBJECT slot of the INGEST is filled by (APPLE REF (INDEF)). The system's current understanding of the input is represented as (INGEST ACTOR (FRED) OBJECT (APPLE REF (INDEF))). There are no more words to read, so the process halts.

The above example gives an idea of the conceptual analysis process. When a word is read, the conceptual structure representing the meaning of that word is added to short term memory. In addition, expectations are created, which provide the processing knowledge necessary to connect up these conceptual structures into a representation of the input as a whole.

3.0 A MINIMAL CONCEPTUAL ANALYZER

Given that one is designing an expectation-driven understander, what mechanism can be used to implement expectations? Much of the system design will depend on this. Following Riesbeck (see Riesbeck (1975)), we chose to implement expectations by using test-action pairs known as requests. If the test of a request is checked and found to be true, then the corresponding actions of the request are executed. Requests are thus a form of production (see Newell (1973)).

In the example of the last section, we mentioned that an expectation for an edible thing would be generated to fill the OBJECT slot of the INGEST structure built by "eat". Implemented as a request, this expectation would look something like this:

TEST: Can a concept representing an edible object be found?

ACTION: Put it in the OBJECT slot of the INGEST concept.

This request is activated when the word "ate" is read or heard. Its test became true in our example above when the meaning of the word "apple" was understood, and its action specified that APPLE was to be taken as the OBJECT of the INGEST. (Usually, we will describe requests even more informally, as something like "If you can find an edible thing, then put it in the OBJECT slot.")

Using this approach, the conceptual analysis process can be most easily described as a special type of production system. Several questions then arise:

(1) What kind of control structure is needed?

(2) What kinds of tests and actions can requests perform?

(3) Where are requests stored, and how are they accessed?

The rest of this section will give preliminary answers to each of these questions in turn.

As the introduction pointed out, we have been very concerned with designing a conceptual analyzer that is flexible. We have formulated the following criterion, in order to make more precise what we mean by "flexibility":

> Criterion of flexibility: A conceptual analyzer must possess the flexibility to alternate between top-down and bottom-up processing as warranted by the situation.

In the framework of conceptual analysis, "top-down" means the ability to utilize expectations if any are active. "Bottom-up" means the ability to hold onto intermediate structures and processing knowledge until some expectations are supplied.

3.1 Control structure

First of all, a conceptual analyzer needs some kind of short term memory, a place to hold the conceptual structures being processed. Without a flexible STM, a conceptual analyzer would be unable to remember substructures which are to be assembled into the final representation of the input. Let us assume that this short term memory is merely a list, called the CONCEPT-LIST or C-LIST. Whenever the processing knowledge of the system, in the form of requests, believes that some conceptual structure is relevant to representing the meaning of the input, that structure is added to the end of this list. Because of this, most processing is focussed on the end of the C-LIST, and so we may think of it as being something like a stack. This kind of flexibility in holding structures is essential if the parser is to function effectively in a bottom-up mode.

Second, a conceptual analyzer needs a data structure to hold the active requests. Without one, of course, the system would never be able to maintain any expectations. In our minimal analyzer (as in previous conceptual analyzers), this is also a list, called the REQUEST-LIST or R-LIST. The system examines the requests on the R-LIST, and if the test of a request is true, its actions are executed. This process is called request consideration.

The control structure of a minimal conceptual analysis algorithm is straightforward, exactly the same as that of previous conceptual analyzers:

(1) Get the next lexical item (word or idiomatic phrase) from the input, reading from left to right; If none, the process terminates;

(2) Load (activate) the requests associated with the new item into the R-LIST;

(3) Consider the active requests in the R-LIST;

(4) Loop back to the beginning (step 1).

3.2 Tests and actions of requests

As was pointed out in the introduction, it is very important that the right vocabulary be devised to express the processing knowledge the system has. Given that we are using requests to embody that processing knowledge, this problem becomes one of choosing the right set of tests and actions that requests may perform.

There are two general classes of tests which requests may perform:

(1) Test for the occurrence of a particular word or phrase;

(2) Examine conceptual structures on the C-LIST, testing for certain semantic or ordering properties.

Requests with the first kind of test are called lexical requests, while those with the second kind are called conceptual requests (or usually, just requests). A more precise classification will be presented in section 6.5, as well as numerous examples of exactly what things requests should test for.

A request may perform any of the following actions should its test become true:

(1) Add a conceptual structure to the C-LIST;

(2) Fill a slot in a conceptual structure with some other structure;

(3) Activate other requests, i.e. add them to the R-LIST;

(4) De-activate requests, including in particular themselves.

There are several variants of these basic actions which will be analyzed when they are discussed in turn in section 6.

In practice, it seems convenient for requests to be more like the LISP 'conditional' than traditional productions. That is, a request should consist of an ordered set of tests controlling branching to mutually exclusive actions.[2]

3.3 Storing and accessing requests

In previous conceptual analyzers, requests were often simply organized under the particular words for which they are useful, that is, requests were stored under words in some kind of dictionary. We have continued to use that method. For example, the request created when reading "an" would simply be stored under the word "an", and activated for use whenever that word is read. This is one way of organizing requests so that only those relevant to the current situation are active in the system.[3]

----

(2) This is particularly useful in allowing a request to notice when it is no longer appropriate and to then remove itself from active status. We can accomplish this by making one of the tests in the request look for clues that the request is no longer appopriate. Should that test become true, the associated action would simply have the request de-activate itself.

(3) Clearly, not all analysis expectations should be implemented as requests associated directly with words. The question of what other structures or mechanisms might be used to organize expectations in memory, and to access them, is a topic of current research.

4.0 AN EXAMPLE

Let's see, in detail, how the minimal conceptual analyzer we have just described might work on another simple example, the statement "Fred gave Sally a book." (The numbers to the left of each step traced below refer back to the steps in the loop of the control structure sketched in section 3.1.)

(1) Read the first word, "Fred".

(2) Activate the requests in the dictionary entry for "Fred". There is only one, and it looks something like this:

```
REQUEST --
 TEST: T
 ACTIONS: Add the structure (PP CLASS (HUMAN) NAME (FRED))
          to the C-LIST
```

The symbol "T" in the test of this request is always true, so that the action should always be performed. When activating this request, the system assigns it the name REQ0.

(3) The active requests are considered. There is only one, REQ0, and it has a true test, so its action is performed. Now, C-LIST = CON1, where CON1 = (PP CLASS (HUMAN) NAME (FRED)).

(1) The word "gave" is read.

(2) Activate the requests in the dictionary entry for "gave". For the purposes of this example, there is just one request in this entry:

```
REQUEST --
 TEST: T
 ACTIONS: Add the structure
          (ATRANS ACTOR (NIL) OBJECT (NIL)
                  TO (NIL) FROM (NIL)
                  TIME (PAST))
          to the C-LIST

          Activate the request
          REQUEST --
           TEST: Can you find a human on the C-LIST
                 preceding the ATRANS structure?
           ACTIONS: Put it in the ACTOR and FROM slots
                    of the ATRANS
```

```
Activate the request
REQUEST --
 TEST: Can you find a human on the C-LIST
       following the ATRANS structure?
 ACTIONS: Put it in the TO slot of the ATRANS

Activate the request
REQUEST --
 TEST: Can you find a physical object on the
       C-LIST following the ATRANS structure?
 ACTIONS: Put it in the OBJECT slot of the ATRANS
```

When activating this request, the system assigns it the name REQ1.

(3) Consider the active requests. There is only one, REQ1, its test is true and so its actions are performed. First, it adds a new structure to the end of the C-LIST. So now C-LIST = (CON1 CON2), where CON2 is the ATRANS structure which represents the meaning of "gave". Then, three new requests are activated, which strive to fill the gaps in the ATRANS structure. The first of these requests strives to fill the ACTOR slot of the ATRANS. When it is activated, the system assigns it the name REQ2. The next request strives to fill the TO slot of the ATRANS. When it is activated, the system assigns it the name REQ3. The last new request strives to fill the OBJECT slot of the ATRANS. When it is activated, the system assigns it the name REQ4.

(3) The system now considers these new requests, and REQ2 is found to have a true test. The action of REQ2 is executed, and so CON1, representing "Fred", is placed in the ACTOR slot of CON2 (the ATRANS) and removed from the C-LIST. The other two requests, REQ3 and REQ4, do not have true tests, so their actions are not executed. However, they do remain as active expectations on the R-LIST.

(1) Read the next word, "Sally".

(2) Activate the requests from the dictionary entry of "Sally". There is only one, and it is almost exactly like the one for "Fred":

```
REQUEST --
 TEST: T
 ACTIONS: Add the structure (PP CLASS (HUMAN) NAME (SALLY))
          to the C-LIST
```

When activating this request, the system assigns it the name REQ5.

(3) Consider the active requests. REQ5 is found to have a true test, so its action is performed. Now C-LIST = (CON2 CON3), where CON3 is the structure representing "Sally".

REQ3, which is still active, now also has a true test, and so CON3 is used to fill the TO slot (recipient) of CON2, the ATRANS structure. REQ4 still does not have a true test.

(1) Read the next word, "a".

(2) Activate the requests from the dictionary entry of the current word. There is only one:

```
REQUEST --
 TEST: Has a new structure been added to the
       end of the C-LIST?
 ACTIONS: Mark it as an indefinite reference
```

When activating this request, the system assigns it the name REQ6.

(3) Consider the active requests. Neither REQ4 nor REQ6, the only active requests on the R-LIST, have true tests. So, nothing happens.

(1) Read the next word, "book".

(2) Activate the requests from the dictionary entry of the latest word. In this case, there is only one request:

```
REQUEST --
 TEST: T
 ACTIONS: Add the structure
          (PP CLASS (PHYSICAL-OBJECT) TYPE (BOOK))
          to the C-LIST
```

When activating this request, the system assigns it the name REQ7.

(3) Consider the remaining active requests. REQ7 has a true test, so its action is performed: CON4 is added to the C-LIST, where CON4 is the structure which represents "book". REQ6 now has a true test, so CON4 is marked as an indefinite reference, like this: (PP CLASS (PHYSICAL-OBJECT) TYPE (BOOK) REF (INDEF)). REQ4 also has a true test, so CON4 is used to fill the object slot of CON2, the ATRANS structure.

(1) There are no more words in the input. The analysis halts, with the following final result representing the input:

```
(ATRANS ACTOR (PP CLASS (HUMAN) NAME (FRED))
        OBJECT (PP CLASS (PHYSICAL-OBJECT)
                   TYPE (BOOK) REF (INDEF))
        TO (PP CLASS (HUMAN) NAME (SALLY))
        FROM (PP CLASS (HUMAN) NAME (FRED))
        TIME (PAST))
```

5.0 PREVIOUS WORK AND A CRITIQUE

Our work has been greatly influenced by previous theories of conceptual analysis and the programs that implemented them. The theory has its roots in the paper of Schank et al. (1970), but the first successful conceptual analyzer was written by Chris Riesbeck, and is described in detail by Riesbeck (1975). The idea of implementing expectations as requests is derived from his work. A later analyzer was Riesbeck's ELI (English Language Interpreter), which is described in Riesbeck and Schank (1976). ELI is a word-based conceptual analyzer of the same general type as we have been describing. Listed under each word that the system has in its vocabulary is the set of requests which implement the expectations a person has upon reading that word. Our ideas were developed in part through experience with this program, and with the problems that arose in its use. ELI's capabilities were expanded by the addition of the NGP program to handle complex noun groups and relative subclauses. NGP was written by Anatole Gershman and is described in Gershman (1977).

The main problem that we found with ELI is simply that it was too top-down. This extremely top-down orientation was a purposeful element of the theory. Unfortunately, we found it to be inadequate for analyzing the kinds of complex input found in ordinary newspaper stories. The next two sections describe in detail the theoretical problems that we identified. The third section describes Gershman's NGP, which solved some of these problems. The fourth discusses some general implications.

5.1 ELI's use of top-down constraints

In the introduction of this paper, we claimed that expectations could be used not only to connect structures, but also to perform word-sense disambiguation. ELI is an attempt to show how requests can be organized to accomplish that task. In ELI, no request listed under a word is accepted by the system for use in processing unless its potential actions include the building of a structure that would make the test of some already active request come true. That is, a request from the dictionary is activated only if the structure it would build satisfies an already existing expectation.

Let's examine exactly how this process works. The tests of the already active requests exist in the form of predicates or combinations of predicates. Call the set of these predicates P. When a word is read by the system, ELI checks each request from its dictionary entry in the following way. First, it finds the structure that the proposed request would build if its test became true. Then, it checks that structure with each of the predicates from the set P. If any of them returns the value TRUE, the new request is activated for potential use. Otherwise, it is discarded. In this way, ELI automatically chooses among the requests associated with a word, and only selects those that its current expectations indicate might be relevant.

Unfortunately, this disambiguation mechanism makes ELI inflexibly top-down. The system cannot use any processing knowledge (in the form of new requests) that it cannot tie to its prior expectations. This necessitates starting the conceptual analysis process with some standard initiating requests at the beginning of each new sentence. However, in the course of processing later clauses, these initiating requests are either inappropriate or unavailable. For example, consider the following sentence:

> A Liberian tanker ran aground off Nantucket Island yesterday, the Coast Guard said.

In attempting to analyze this sentence, ELI has difficulty with the second clause, because there are no expectations for anything beyond the first. The problem arises because ELI's disambiguation algorithm does not activate a request unless it will build a structure satisfying an outstanding expectation. Hence if, as in this example, there are no outstanding expectations, no requests can be activated. The requests associated with the words "the Coast Guard said," are simply ignored.

This inflexibility also makes it impossible for ELI to postpone making decisions. For example, ELI could choose the correct sense of "pot" in each of the following sentences:

(1) John broke the pot.

(2) John smoked the pot.

In the first sentence, "broke" creates an expectation for something which can be broken. The disambiguation algorithm uses this expectation to activate the request which builds the "container" sense of "pot", and reject the one which builds the "marijuana" sense. However, in the second sentence, "smoked" creates an expectation for something which can be so ingested, which is then used to choose the "marijuana" sense.

But ELI would not be able to disambiguate "pot" in the passive forms of these sentences:

(1′) The pot was broken by John.

(2′) The pot was smoked by John.

Its failure here is due to the fact that at the time the word "pot" is read, there are no semantic expectations which can be used to choose among the differing word senses. Because ELI′s disambiguation algorithm only uses prior expectations, it cannot postpone deciding among the requests associated with "pot" until later in the sentence, when the proper semantic expectations are available to perform the disambiguation. Nor does it save any discarded possibilities in case they should prove useful later on.

5.2 ELI′s Short term memory

We mentioned above that ELI requires a set of standard initiating requests at the start of a sentence. This necessity is also related to the structure of the program′s short term memory. ELI′s analysis process starts with a three item short term memory. One of these slots is the place where all new conceptual structures are placed when built, called the <u>input gap</u>. The other two are essentially the "subject" and the "top-level concept" slots. These two slots are filled by two standard initiating requests which take structures from the input gap. Once these two slots are filled, they cannot be changed except by use of arbitrary LISP code in the requests. This essentially leaves ELI with a working short term memory capacity of only one item, which is insufficient for noun groups and many multi-clause constructions. For example, consider again the sentence:

A Liberian tanker ran aground off Nantucket Island yesterday, the Coast Guard said.

Even if the requests associated with the words of the second clause ("the Coast Guard said") were accepted for processing, the limitations on short term memory would prevent a correct analysis. For, the representation of "Coast Guard" would be placed in the one remaining STM slot, but then written over and lost when the representation for "said" was placed there in the next cycle of the system.

5.3 Noun groups

As we pointed out above, ELI did not process noun groups or relative subclauses itself. These were handled by Gershman's NGP program. NGP necessarily has a far more flexible short term memory, consisting of a modified stack. To understand why this more flexible short term memory is necessary, consider trying to analyze a noun group like "blue car seat". If only a one-item working short term memory were available, then the representation for "car" would be overwritten by the representation for "seat", and hence lost. However, NGP's use of a modified stack STM can handle this quite easily.

When analyzing a word in some noun group, NGP creates a new node, and then pushes it onto the stack which constitutes the program's short term memory. The conceptual structure associated with the word is placed in that node, and any new requests are activated by being put on the property list of that node. Gershman makes a distinction between "forward" requests, which look one node to the right, and "backward" requests, which look one node to the left. In any one cycle of NGP, the only requests which are considered are the "backward" requests associated with the top node in the stack, and the "forward" requests associated with the next-to-the-top node.

5.4 Criterion of flexibility

Our experiences with the problems that a too inflexibly top-down control structure can cause led us to formulate the criterion of flexibility which was stated in section 3.0:

> A conceptual analyzer must possess the flexibility to alternate between top-down and bottom-up processing as warranted by the situation.

This means that the analyzer must be able to use expectations when they are available. But if no helpful expectations are available, the system must also be able to accept and save concepts and requests from the input until some later time when they can be used. In our conceptual analyzer as so far described the C-LIST and R-LIST provide this flexibility. In section 9 we will discuss some solutions to the problem of word-sense ambiguity which do not violate the criterion of flexibility.

6.0 REQUESTS IN MORE DETAIL

The purpose of this section is to provide further details about requests. We will describe the kinds of tests and actions that we found to be necessary to accomplish conceptual analysis, discuss why they are necessary, and the situations in which they are useful.

6.1 Request actions: adding structures to C-LIST

Let us reiterate the actions which a request may perform:

(1) Add a conceptual structure to the C-LIST;

(2) Fill a gap (i.e., slot) in a conceptual structure with some other structure;

(3) Activate other requests;

(4) De-activate requests, including in particular themselves.

There is no restriction that a request perform only one of these actions. However, the convention that a request may only add one structure to the C-LIST is important for the purpose of designing word-sense disambiguation algorithms, and we have adhered to it. (This will be explained in section 9.) As stated before, structures are always added to the end of the C-LIST.

6.2 Gap-filling requests

Conceptual structures represented as case frames are connected when one fills a gap (i.e., slot) in the other, or both fill a gap in some larger structure. Requests which fill gaps can perhaps be thought of as moving conceptual structures from one place to another. This perspective suggests that embedded conceptual objects are removed from public view, presumably by actually removing them from the C-LIST. This is how our program currently works. This decision was motivated by our view of the C-LIST as literally a kind of short term memory, with limitations on its storage capacity. A positive side-effect of this is to speed up search of the C-LIST.[4]

We can characterize gap-filling requests in roughly two ways. Either the request knows which gap it's trying to fill, and is looking for a filler, or it knows the filler, and is looking for the proper gap. Requests of the first type are looking for some structure in order to embed it in another. We have already seen numerous examples of this type of gap-filling request in the previous examples. These requests may arise in two ways:

(1) They may be activated by the request which built the conceptual structure containing the gap they strive to fill.

(2) They may be brought in by function words (e.g. "to").

Gap-filling requests arising from function words are particularly important in that if a request activated by a function word seeks to fill some gap with a structure, it must be allowed to do so. If the gap is already filled by some other structure, that structure should be returned to the C-LIST. For example, consider the sentence "John gave Mary to the Sheik of Abracadabra," (see Wilks (1975)). After reading "John gave Mary ..." our conceptual analyzer would assume Mary to be the recipient of the giving action, and put together a representational structure something like (ATRANS ACTOR (JOHN) OBJECT (NIL) TO (MARY) FROM (JOHN)). However, the function word "to" clearly marks the Sheik as the recipient, and that explicit marking has high priority. Hence, the analyzer would change its representation to

----

(4) One could instead simply mark objects which have been used to fill gaps, perhaps with the information as to where they are embedded, and leave them on the C-LIST. This latter method is more flexible, in that other requests can selectively ignore embedded objects or not, depending on the purpose.

(ATRANS ACTOR (JOHN) OBJECT (NIL) TO (SHEIK) FROM (JOHN)). The concept MARY would be returned to the C-LIST, and subsequently picked up as the OBJECT of the ATRANS.

As we pointed out above, there is a second type of gap-filling request which knows the filler, and looks for some structure in which to embed it. These usually arise from words that appear in noun groups, particularly adjectives. For example, the word "red" has an associated request which looks for the representation of a physical object, in order to fill its COLOR slot with the structure RED. This kind of gap-filling request also arises from words describing time and place settings. For example, in the sentence "Yesterday, Fred bought a car," the word "yesterday" has an associated request which looks for a concept, in order to fill its TIME slot.

Once a gap in some structure is filled, all other requests which seek to fill it should be removed. That can be handled in two ways. The simplest is to have requests which seek to fill some particular gap test whether it has already been filled, and if so then remove themselves. This is how our current implementation operates. The other possibility is to use two-way pointers between gaps in conceptual structures and the requests which strive to fill them, and to place in the control structure a procedure which uses this information to remove unnecessary requests. Riesbeck's ELI uses this approach.

6.3 Activating other requests

Two major purposes motivate our use of requests to activate other requests. First, as we pointed out above, requests that add some conceptual structure to short-term memory will often activate requests which strive to fill gaps in that structure.

Second, some requests <u>change the sense of some word in some local context</u> (see Riesbeck and Schank (1976)). This can be accomplished by activating a request, which, if it sees the appropriate word in the input, will add some requests to the dictionary entry of that word. Function words are an important instance of this notion of locally changing word sense, since in many cases they function only with respect to a particular construction. Take the example of the word "of". The relationship it denotes depends on the construction in which it appears. One such construction includes examples such as "five yards of cloth", or "ten pounds of marijuana". A good way to handle this construction is to have a request which is associated with the "unit" sense of words such as "yards" and "pounds". This request is activated by the request which adds the "unit" structure to the C-LIST. When active, the request checks to determine whether or not the next word is "of". If it is, then the request activates another request which looks for a physical object, and when found attaches the description created by "five yards" or "ten pounds" to it.

In general, locally changing the sense of some word in this manner does not involve throwing out whatever requests are indexed directly under the word, just augmenting them. In this way, if the newly activated requests should fail to be applicable, then all of the usual requests associated with the word are still available to continue processing. However, the specially added request needs to be guaranteed priority over the usual requests associated with the word.

6.4 De-activating requests

The final request action is the ability to de-activate requests. One important use of this action is to have requests de-activate themselves under certain circumstances. In this way, a request which notices that it is no longer appropriate can remove itself before fouling up the analysis. For example, consider the request for locally extending the senses of the word "of", described in the last section. If the next word is not "of", then this request must de-activate itself. Otherwise, if "of" occurs later in the input, the request would change its meaning in a way which is no longer appropriate.

Another situation in which this action is useful occurs when there are several requests which represent competing hypotheses about the meaning of a word. The request which represents the correct meaning (however chosen), must de-activate the other requests. (We will return to this in section 9, which discusses the problem of ambiguity.)

6.5 Tests of requests

As we discussed in section 3 above, the tests of requests examine either the actual words of the input text, or the objects on the C-LIST. We have at this point distinguished two variants of the latter:

(1) Searching for the occurrence of some structure on the C-LIST with the correct properties;

(2) Testing whether or not certain gaps in some structure are filled.

Clearly the properties with which tests of type 1 are concerned will often be conceptual or semantic. For instance, one might write a test like "is there a PP on the C-LIST?", or "is there a PP on the C-LIST which is a higher animate?". We have seen these sorts of tests in the examples discussed previously. But, these tests may also have constraints on <u>where</u> in the C-LIST they should look, such as, for example, an instruction to look only on the end of the C-LIST, or to search only preceding or following some other object on that list. Such constraints are one way of utilizing the structural information derived from word order. In other words, these constraints, which can be expressed by predicates in the test, embody expectations related to sentence structure rather than content.

These expectations which embody syntactic knowledge are necessary whenever there are several gaps in a representational structure which have the same semantic requirements. For example, both the ACTOR and the TO slots of an ATRANS can appropriately be filled by a "higher animate". Syntactic knowledge must then be used to decide which of several appropriate gaps a structure should fill. Our conception of syntax is more fully explained in section 8.1.

Two predicates were found useful in constraining where on the C-LIST to look: (1) a predicate which tests whether or not one structure <u>precedes</u> another on the C-LIST; and (2) a predicate which tests whether or not one structure <u>follows</u> another on the C-LIST. We have also already shown examples of this sort of test, in section 3 above. In the example given there, the ATRANS sense of the word "give" activated the following request:

```
REQUEST --

  TEST: Can you find a human on the C-LIST
        preceding the ATRANS structure?

  ACTIONS: Put it in the ACTOR and FROM slots
           of the ATRANS
```

The type 2 test in the above scheme reflects another possible method for utilizing word order information. Rather than depending on the order of objects in the C-LIST, the determination of which of several unfilled gaps some candidate structure should fill could be based in part on constraints in the order in which those gaps should be filled. These constraints are expressed by having tests which check to see whether or not some gap is filled. For example, suppose we wanted to write a request to fill the TO slot of an ATRANS arising from the word "give". Because the same semantic constraints apply to the ACTOR and TO slots, we need a test to distinguish between them. Since, in an active sentence, the ACTOR will be seen before the recipient is seen, the following request will insure that the TO slot is not incorrectly filled with a structure that ought to fill the ACTOR slot:

```
REQUEST --

  TEST: Can you find a human on the C-LIST, and is the ACTOR
        slot already filled?

  ACTIONS: Put it in the TO slot of the ATRANS.
```

6.6 Searching the C-LIST

An ordering problem arises in conceptual analysis when the test of a conceptual-level request is searching the C-LIST for some object with certain properties. If more than one item on the C-LIST satisfies the test, how can the system choose between them? Let's examine the following fragment: "The girl Fred saw in the park ..." The request activated by reading "saw" activates another request looking for an animate actor. But there are two possibilities, "the girl" and "Fred". In cases like this, we use a recency rule to select from among several possibilities. That is, when scanning the C-LIST looking for some object which satisfies certain predicates, the system should look at the more recently added objects first.

7.0 CONTROL STRUCTURE IN MORE DETAIL

So far, we have not elaborated much on the control structure described in section 3. The purpose of this section is to provide further details about the control structure of a conceptual analysis algorithm, and to describe the reasons which motivate that control structure.

7.1 Organizing requests: the recency rule

A large part of the theory of conceptual analysis is the problem of request organization, which is related to controlling how and when requests get considered.[5] We have supposed, rather simplistically,

----

(5) Similar control questions arise in all production systems. For a good overview, see Davis and King (1975). The central problem is determining which control strategies are appropriate to which domains.

that active requests are held in some list, the R-LIST. Now consider the following example sentence: "Saul ate a big red apple." Our intuition is that the adjectives preceding the word "apple" have a chance to modify the concept "APPLE" before the resulting concept is used to fill the INGEST frame added to the C-LIST by "ate". It appears that the requests activated by those adjectives are considered before the requests activated when the INGEST structure was added to the C-LIST. Examples like this, and others as well, have led us to adopt a rule of <u>request recency</u>. (Rules of this sort are common in production systems.) This rule states that requests are considered in reverse order of their activation, from most recent to least. In addition to guaranteeing that the analysis proceeds according to our intuitions, this is a good heuristic principle, since newer requests represent newer, and so possibly better, information about what might be going on than older requests.

Let's look at another example: "Fred told John Bill hit Mary." When "told" is read, a request is activated which adds a conceptual structure to the C-LIST, roughly something like (MTRANS ACTOR (NIL) MOBJECT (NIL) TO (NIL)). In addition, requests are activated which try to fill the empty slots in this frame. One of these requests is looking for a concept to put in the MOBJECT slot. Then, when "hit" is read, a request is activated, which adds the conceptual structure (PROPEL ACTOR (NIL) OBJECT (NIL)) to the C-LIST.

This request also activates some new requests to try and fill the empty slots in the PROPEL frame. Even though the system has already commenced scanning the list of active requests, these new requests are now the most recent. The process could simply make a note that new requests have been activated, and continue checking the older requests. This corresponds roughly to a "breadth first" consideration of requests. A strict interpretation of the recency rule, however, leads us to postulate that request consideration is more "depth first". That is, newly activated requests, being the most recent, are considered immediately. Going back to our example, this guarantees that the requests which strive to fill gaps in the PROPEL frame are considered before the requests which strive to fill gaps in the MTRANS frame. Hence, "Bill" is used to fill the ACTOR slot of the PROPEL before the resulting structure is used to fill the MOBJECT slot of the MTRANS.

In many cases, more than one request is activated at a time, and hence these requests have the same recency. Since requests are considered in order of recency, requests activated at the same time should be clearly marked as such. A set of requests activated at the same time, and hence having the same recency, is called a request pool. Thus, in order to implement the recency rule, our program uses an ordered list of request pools, each pool containing one or more requests. Section 7.3 will discuss request organization within pools.

7.2 An alternative to the recency rule

The main import of the recency rule seems to be that if a conceptual structure A is going to be used to fill a gap in some other structure B, then before A is embedded in B, A is given a chance to fill its own gaps. Consider again the example of the last section. Before the PROPEL structure is embedded in the MTRANS, "Bill" is used to fill the ACTOR slot of the PROPEL. In processing terms, this effect of the recency rule can be expressed as follows: if requests which assemble and modify lower level structures and requests which assemble these sub-structures into higher level structures are active at the same time, then the former have priority over the latter. Taking this formulation as our main principle of request organization seems to lead to equivalent order of processing, but a completely different algorithm. The most obvious implementation of this principle is as follows:

(1) Collect all requests with true tests, without evaluating any actions.

(2) Order these requests so that if a request has an action which uses some structure to fill a gap, any requests which want to change that structure or fill a slot in it are given first priority.

(3) Execute the requests.

Our program currently uses the recency rule, since it is easier to implement.

7.3 Request organization within pools

The next question is, how are requests organized within pools? A certain amount of hierarchy is clearly necessary, since, as we noted in the previous section, requests specifically added to change the sense of a word in some local context must be guaranteed priority.

In addition to this hierarchical organization, requests are organized within pools according to the actions they perform. For example, suppose that several requests in the same pool all have among their actions adding some structure to the C-LIST, and suppose further that more than one of these has a true test. This is how word-sense ambiguity would manifest itself in a conceptual analyzer, and the problem is to select the correct request. Thus, the algorithm which controls request consideration must not only order the requests, it must sometimes choose among them. If it were unable to make such a choice, it might even prevent any of some set of competing requests from being executed. In section 9, we will return to the question of how the algorithm which controls request consideration might be able to choose among requests in cases of word-sense ambiguity.

7.4 Noun groups

An analyzer implemented along the lines we have described up to now would have great difficulty processing noun groups with more than one noun correctly. The problem is, it is possible that a concept will incorrectly use some intermediate conceptual structure to fill a gap. For example, consider the sentence "George sat on the stairway handrail." The concept representing "sat" presumably has an empty slot for the object upon which the actor was sitting, and a request which tries to fill that slot (let us agree to call it the OBJECT slot). Since a stairway is a perfectly fine object to sit on, the algorithm as described up to this point would allow that request to fill its slot with "stairway", before the entire noun group had been analyzed. So, the problem is to somehow prevent any request striving to fill the OBJECT slot from firing prematurely, or correct the situation if that should happen. It is clear that when analyzing noun groups the control structure must consider requests in some manner different than so far described. Our problem is once again one of controlling request consideration.

To handle this, we use the following procedure for request consideration in noun groups: Consider <u>only</u> the requests associated with each incoming word (i.e. the most recent request pool) until the end of the noun group, when processing returns to normal (i.e. considering all requests starting with the most recent pool). That is, if the program is in this noun group mode, and a new word is input, then <u>only the newly activated requests associated with that word are considered</u>. Prior requests are <u>not</u> considered. The reader should note that use of this algorithm depends on ordering the requests by recency.

Using this changed procedure for noun groups, the entire structure representing "stairway handrail" would be constructed <u>before</u> the request which strives to fill the OBJECT slot of the sitting action could be considered.

This procedure for controlling request consideration has the additional effect of insuring that the head noun of the noun group is discovered before adjectives are attached. For example, consider how this algorithm would analyze the input "blue car seat". First the request associated with the word "blue" would be activated. Such a request would strive to fill the COLOR slot of some physical object to the right. Next, the request associated with the word "car" would add a structure to the C-LIST representing the concept of automobile. However, since the analyzer is in noun group mode, the prior expectation from "blue" would <u>not</u> be considered at this time. Finally, the word "seat" is read, and another structure is added to the C-LIST. Now, only after the structure representing "car seat" is built, would the request associated with "blue" be considered. It would then ascribe the attribute of being colored blue to the seat, not to the car. Under certain circumstances, of course, this could be wrong, e.g. "red stairway railing" might refer to the railing of a red stairway, instead of the red railing of a stairway. We believe that a general solution to problems of this kind will depend on better understanding of the role of memory in parsing. However, as we pointed out above, the process described here does have the virtue that it allows noun groups to be completed before the resulting analysis is used in larger structures. For example, if the phrase "blue car seat" is embedded in the sentence "George sat on the blue car seat," then the analysis would determine that George sat on the seat, not on the car.

In order to use the process we sketched out above, an analyzer must be able to determine when it should be in noun group mode (which is, in some sense, a "careful" mode), and when it should be in normal mode. We have used, with slight modification, Gershman's (1977) heuristics for determining noun group boundaries. Some of these rules depend on conceptual, and some on syntactic, knowledge.

8.0 SOME THEORETICAL ISSUES

In this section, we will address some of the theoretical concerns which motivated our work.

8.1 The role of syntax

In our discussion to this point, we have repeatedly emphasized that the analysis process should use any available knowledge which might be helpful, including knowledge of syntax. The reader may at this point be wondering exactly what role syntactic knowledge has in conceptual analysis, and how it is used. Traditional notions of syntax include ideas like "part of speech" and "phrase marker" in discussing the structure of a sentence. What we would like to claim in this section is that these notions of syntax are inappropriate when attempting to describe and utilize syntactic knowledge in a language analysis process.

To begin with, what is the purpose of syntactic knowledge? Clearly, a major use of syntactic knowledge is to direct the combination of word meanings into utterance meaning when semantic information is not sufficient. For example, in an utterance like "Put the magazine on the plate," it is syntactic knowledge that tells the understander which object is to be placed on top of which. From the point of view of its use in a conceptual analyzer, therefore,

> A large part of syntax is knowledge of how to combine word meanings based on their _positions in the utterance_.

How can this syntactic knowledge be characterized? We seek a specification which takes into account the fact that the point of syntax is its _use_ in the understanding process. We have viewed the process of understanding as one of connecting representational structures, where a connection has been established between structures when one fills a slot in the other, or both fill a larger form. Thus syntactic knowledge is knowledge which specifies <u>where</u> in the utterance some word is to be found whose meaning can be connected (via slot-filling) with the meaning of another word. Of course, we must now specify the notion "position in an utterance".

In section 6.5, we discussed how knowledge about

(1) Relative positions in short term memory, and

(2) The order in which slots in structures should be filled

could be used in the tests of requests. Both of these methods describe position in an utterance by use of <u>relative</u> positional information. The first of these methods, using relative position in short term memory, describes the position of a conceptual structure in <u>direct</u> relation to the other structures it might be connected with via slot-filling. The second method, ordering of the slots to be filled, relates the position of a structure to other structures somewhat more indirectly. In particular, by constraining the order in which slots should be filled, we are relating the fillers of those slots <u>temporally</u>. But, of course, <u>when</u> a structure becomes available is directly related to <u>where</u> it appears in the input, i.e. relative position in short term memory reflects the temporal order of the input. Both methods are essentially ways to utilize word order information.

Another important method relates the position of a conceptual structure to the position of a particular lexical item, i.e. a function word. Function words, including prepositions, post-positions, and affixes, are very important syntactic cues in language analysis.

8.2 Prior syntactic expectations unnecessary

In our discussion of the analysis procedure, we have repeatedly stressed the importance of flexibility in switching between top-down and bottom-up modes of processing. This is necessary, for example, in order to handle long multi-clause constructions. Having determined how to achieve this flexibility, we discovered an interesting side-effect. We found that it was not necessary for the analyzer to have any prior expectations at the start of a sentence, such as for an initial noun-group, or for a complete sentence. Most other parsers, whether syntax based or semantics based, do use such prior expectations at the start of a sentence. While we were at first unsure as to the psychological correctness of the absence of these prior syntactic expectations, we now believe that this is in fact the correct approach.

However, there are some problems that might arise given this approach. How can we explain the weirdness of sentence (1) below, as opposed to sentence (2)?

(1) A plane stuffed with 1500 pounds of marijuana.

(2) A plane was stuffed with 1500 pounds of marijuana.

There is a real feeling of "incompleteness" in sentence (1) which would seem to argue that people <u>do</u> have prior expectations for complete sentences. However, consider how these two sentences would sound following this question:

(3) What crashed?

Following question (3), it is sentence (2) that sounds odd, while sentence (1) seems perfectly at home. Now, one could explain this phenomenon by saying that the expectations to be supplied at the beginning of a sentence change, and that after question (3), a parser is no longer expecting a complete sentence. But of course, question (3) could equally well be followed by any of the following:

(4) A plane carrying 1500 pounds of marijuana crashed.

(5) Nothing crashed.

(6) What are you talking about?

(7) There wasn't any crash.

(8) I didn't hear anything.

So obviously, a parser which depends on an expectation for a complete sentence, could not simply throw it out in the context of a question like (3). If it did, it would then be unable to handle inputs like (4) through (8). Further, these examples make clear that the absence of an expectation for a complete sentence, within the context of a question, cannot be the explanation for the weirdness of (2) in the context following (3). That can only be explained by reference to higher level processes, such as memory search and question answering. However, it seems to us that the weirdness of (1) in a null context is exactly analogous to the weirdness of (2) in the context of (3). And, of course, it can be explained by exactly the same appeal to the properties of higher level processes, <u>which we have seen is already necessary</u>. Therefore, this phenomenon cannot be used as evidence against our contention that prior syntactic expectations for e.g., a complete sentence, are unnecessary.

We would claim, moreover, that the explanation for the symmetrical cases of (1) in a null context and (2) in the context of (3), ought to be the same. In other words, the explanation underlying this particular type of "grammaticality" judgment does not lie in a syntactic phenomenon at all. Rather, it is a phenomenon which should be explained by reference to such higher level processes as memory search and question answering.

8.3 Learning conceptual analysis

Our work has been significantly affected by developmental concerns. A theory of conceptual analysis, if it is to be taken seriously as a psychological theory, must ultimately show how the process of conceptual analysis can be learned, starting from some minimal base of knowledge and procedures (see Schank and Selfridge (1977) and Selfridge (1979)). It is not feasible to discuss here what that base might be, or what the learning mechanisms are. But it is appropriate to point out how these considerations have affected the work described here. For instance, our goal of a simple control structure is motivated by more than pragmatic considerations. The less additional structure that must be proposed, the more the control structure resembles a very minimal computational device. That in turn makes the problem of learning conceptual analysis more a problem of learning the proper requests, which might lead to more attractive developmental theories. For example, if one asks a young child to "Put the table on the doll," he or she might well respond by putting the doll on the table. (See, for example, Hoogenraad et al. (1978).) It seems as if the child actually understands the utterance as having that (quite plausible) meaning. Yet, an adult clearly understands the actual meaning, despite its absurdity. The theory of conceptual analysis suggests that the difference displayed here between adult and child language understanding might be a difference in the sophistication of the expectations (i.e., requests) which are used to fill the gaps in the case structure which represents the meaning of "put". That is, the tests of the requests possessed by the child are almost completely semantics based, whereas those of the adult are more syntactically sophisticated. It is quite plausible now to hypothesize processes whereby those requests can be changed by experience to reflect the maturation which has occurred. A precise specification of such processes would constitute an important part of a theory of language acquisition.

8.4 Intelligent error correction

Recently, some attention has been focussed on the idea of deterministic "wait and see" syntactic parsing, without backup (see Marcus (1975)). Determinism is guaranteed by restricting the analysis process so that no structure can be built, but then later discarded. This notion has been contrasted with the non-deterministic "guess and then backup" method typified by ATN parsers. While we are in complete agreement with the idea that it is undesirable to use blind backup as an integral part of the analysis process, we believe that Marcus' notions of determinism have ignored the possiblities of what might be called "intelligent error correction". For example, let's look at the following pair of sentences:

(1) John gave Mary a book.

(2) John gave Mary a kiss.

In the first, "gave" should add the following structure to the C-LIST: (ATRANS ACTOR (NIL) OBJECT (NIL) TO (POSSESSION-OF (NIL)) FROM (POSSESSION-OF (NIL)). In the second, "gave" is merely a dummy verb, and this structure is completely inappropriate. However, it turns out that there are two ways to handle this problem in a conceptual analyzer. The first approach is exactly the analogue in conceptual analysis of what Marcus would call "wait and see", namely not building any top-level structure until either "book" or "kiss" is read. The second approach is "intelligent error correction". Using this method, the ATRANS structure is built in both cases, and the analyzer proceeds to fill the empty gaps with appropriate substructures. However, an additional request is activated which essentially says "If you see another complete concept, i.e. action or state, rather than something that can fill the object slot of an ATRANS, then that is the actual top-level structure. Remove the ATRANS from the C-LIST, and fill the empty gaps in the new structure with the substructures which can be found in the ATRANS." Using either of these methods, the sub-structures which have been assembled along the way, such as the representations for "John" and for "Mary", are still available, and this after all is the key to avoiding back-up. Using intelligent error correction might be the better approach in those cases where the initial hypothesis is almost always correct (see Gershman (1979)).

9.0 WORD-SENSE DISAMBIGUATION

As we pointed out in section 7.3 above, in conceptual analysis word-sense ambiguity manifests itself when two or more requests of equal recency (i.e., in the same pool, and therefore derived from the same word) have among their actions adding structures to the C-LIST. How can we insure that the structure representing the correct meaning "in context" can be chosen? What we describe in this section, of course, does not constitute anything like a complete answer to the problem of word-sense ambiguity. However, we do present some improved mechanisms for disambiguation in conceptual analysis, and perhaps clarify what the possibilities are.

9.1 Using requests to check the context

There are essentially two ways to handle word-sense disambiguation. Either the structures which represent alternate hypotheses check the context in order to decide whether they are appropriate, or the context checks the structures in some way, or both. In conceptual analysis terms, this means that there are two ways to accomplish the task of request selection in order to handle ambiguity. The first method is to have the requests check the context. This is accomplished by having the tests of the requests check as much as possible for clues as to whether or not the structure they would add to the C-LIST is appropriate. These tests could be at either a lexical level (i.e. checking for specific words or phrases), or at a conceptual level (for instance checking other structures on the C-LIST). For example, to handle the difference between "John left the restaurant" (PTRANS) and "John left a tip" (ATRANS), the requests associated with the word "left" could check to see whether something that could be a location, or something that could be an object, is in the input.

9.2 Using the context to check the requests

The second way to perform request selection is to have some more general algorithm which would use information in the context to examine the requests, and choose among them. In other words, we need some general algorithm which uses the conceptual context to choose from among the possible structures being offered. The traditional method for doing this is to choose the structure which can be connected, or perhaps connected most "richly", with other structures the system has in its short term memory. (An early incarnation of this idea can be seen in the use of "selection restrictions" to rule out certain combinations of meanings in Katz and Fodor (1963).) In conceptual analysis, the most obvious application of this idea involves choosing that structure which can be used to fill gaps in other structures on the C-LIST, or that can use other structures on the C-LIST to fill its own gaps, or both.[6] The example of the last paragraph can also be handled by the method of choosing a structure which can use other structures on the C-LIST to fill gaps, over a competing structure which cannot. This is because the ATRANS sense of "left" can make use of the structure associated with "tip" to fill its

----

(6) This is clearly not the only kind of "connectedness" or "coherence" which can cause us to favor one sense over another. In particular, the structures of the context which are used to check some potential structures need not be directly derived from the input, that is, should also include structures derived by inferential memory procedures.

OBJECT slot, while the PTRANS sense can use the structure associated with "restaurant" to fill its FROM slot.

9.3 Previous work

Neither of these two methods is new, of course. Riesbeck's ELI has employed the first method, i.e. having requests test the context to determine their applicability, with some success.[7] Wilks' preference semantics scheme (see Wilks (1976)) includes probably the most highly developed mechanism of the second kind, i.e. searching for connections (see also Hayes (1977)). ELI also uses this general method, but at request activation time, not consideration time as we are advocating here. This means that when examining a set of requests, the request selection algorithm has only one chance, and can only make use of the information the system has at that time.

9.4 A possible disambiguation algorithm[8]

Having requests check context requires no special addition to the control structure of a conceptual analyzer, since the work is done by the requests. Having context check requests, however, requires augmenting the mechanism which controls request consideration. We will make use of only those obvious connections between structures which arise when one can fill a gap in the other, as discussed in

----

(7) Rieger and Small are currently pursuing an interesting line of research which depends heavily on this mechanism. See Small (1978).

(8) This aspect of our analyzer has not been implemented.

section 9.2. The algorithm should be flexible, and that means that if there is not enough evidence to make a selection, the system must be able to wait and try again later.[9]

We will assume that requests are organized within pools according to the classification of section 7.3 above, and will concern ourselves with those requests which strive to add some structure to the C-LIST. The mechanism for controlling these requests within each pool must be applied any time the requests are considered, and should look something like this:

> (1) Collect those requests which have true tests.
>
> (2) If more than one has a true test, then check the structures which these add to the C-LIST, to see if they can either fill a gap in some other structure already on the C-LIST, or can use some other structure to fill one of their gaps.

At this point, three things can happen: either none of the requests succeed in this check for connections, only one succeeds, or several succeed. The first case is easy to handle:

> (3a) If none of the requests succeed in the check for connections (step 2), then don't let any of them go off (i.e., postpone making a decision).

The second case is also relatively easy, although complexities arise when we consider the possibility that what seems like the only choice at one point, may turn out to be a poor choice later on:

----

(9) The ability to postpone choosing a word sense raises the issues of how long to wait, whether there are default meanings, and so forth. (The discussions of section 8.4 are relevant here.) Nevertheless it seems clear that the ability to postpone choosing, for at least a short time, is necessary in many instances.

(3b) If only one of the requests succeeds in the check for connections (step 2), then use that (i.e., perform its actions), and de-activate the rest.

The third case is quite difficult:

(3c) Else, don't let any of those remaining go off; now a more complex decision needs to be made.

This third case probably requires postponing a decision, until one or the other structure is found to be the "best" connected. In Wilks' system, the "best connected" is the "most connected", and this is certainly one possible measure. However, Wilks' analyzer performs this comparison only at what are essentially clause boundaries, and does not rule out any possibilities until then. In this third case where several requests succeed in the check for connections, it seems plausible that those which do not could be de-activated immediately, thus reducing fairly quickly the set of viable alternatives.

The use of some disambiguation process as outlined above would place new requirements on the control structure of a conceptual analyzer. In particular, we have often depended on a fairly consistent correspondence between the order of input of some word, and the location in the C-LIST of any associated structure. But, since the requests which add structures to the C-LIST could be prevented from firing immediately by the disambiguation process, such a correspondence is no longer likely. We could get around this problem by writing more sophisticated predicates that, when testing some structure on the C-LIST, would check to see when the request which added it was activated. More intuitively, we could change our control and data structures a bit to guarantee the relationship between order of input and order on the C-LIST. The easiest way to do that would be as follows: whenever a pool of requests is activated, a new empty node would be added to the C-LIST. Should a request in that pool add some structure to the C-LIST, it would add it in the corresponding node, not at the end as previously described. Thus, the C-LIST would then consist of a list of such nodes, some with structures, and some empty. An empty node would often have associated requests striving to add structures to the C-LIST in that position. Such a data structure, linking the C-LIST and request pools explicitly, resembles quite closely the one used in Gershman's NGP. However, his system uses the data structure quite differently in controlling request consideration.

10.0 THE PROGRAM

Our ideas on conceptual analysis have been implemented in a program which has proven to be a robust system capable of analyzing complex input in many domains. It has been used in conjunction with a version of Cullingford's Script Applier Mechanism (SAM; see Cullingford (1978)), and as part of Lehnert and Burstein's Object Primitive Understanding System (OPUS; see Lehnert and Burstein (1979)). It is currently being adapted for use as a front end to Kolodner's conceptually oriented natural language data base (CYRUS; see Kolodner (1978)), and for use in a conversational program. The system is capable of automatically handling passive constructions. The routines for accomplishing this were devised by Rich Cullingford. To give an idea of the range of the system, the rest of this section will be devoted to presenting some examples. All of the examples in this section are actual input to and output from the program.

The following pair of questions, "Who is Cyrus Vance?" and "Who is Vance meeting?", are essentially the same except for the addition of one word to the latter. Yet they differ tremendously in meaning.

```
*(CA '(WHO IS CYRUS VANCE?))

(EQUIV ACTOR
       (PP PPCLASS (HUMAN) LAST-NAME (VANCE) FIRST-NAME (CYRUS))
       OBJECT
       (PP PPCLASS (HI-ANIM) SPEC (*?*))
       TIME (*NOW*)
       MODE (NIL))

*(CA '(WHO IS VANCE MEETING?))

($MEETING ACTOR
          (PP PPCLASS (HUMAN) LAST-NAME (VANCE) FIRST-NAME (NIL))
          OBJECT
          (PP PPCLASS (HI-ANIM) SPEC (*?*))
          TIME (*NOW*)
          MODE (NIL))
```

The following pair of questions presents a problem similar to the previous pair. That is, they look almost the same but differ greatly in meaning. However, in this case the reason for the difference in meaning is not due simply to the addition of an extra word, but to more subtle semantic differences.

```
*(CA '(WHERE DID VANCE GO LAST WEEK?))

(PTRANS ACTOR
        (PP PPCLASS (HUMAN) LAST-NAME (VANCE) FIRST-NAME (NIL))
        OBJECT
        (PP PPCLASS (HUMAN) LAST-NAME (VANCE) FIRST-NAME (NIL))
        TO
        (LOC SPEC (*?*))
        TIME
        (TIME BEFORE
              (*NOW*)
              AMOUNT
              (PP PPCLASS (UNIT) TYPE (WEEK) NUMBER (NUMBER VAL (*1*))))
        MODE (NIL))
```

```
*(CA '(WHERE DID VANCE GO TO COLLEGE?))

($COLLEGE ACTOR
         (PP PPCLASS (HUMAN) LAST-NAME (VANCE) FIRST-NAME (NIL))
         TIME (*PAST*)
         MODE (NIL)
         PLACE
         (LOC SPEC (*?*)))
```

The following example shows that the analysis process we have described is capable of handling long multiple-clause inputs. In this particular case, the input is not analyzed into a single connected concept, since the knowledge that a plane crash can lead to the death of an occupant of the plane is not to be found under the word "kill". Higher level processes, such as a script applier, will have the necessary knowledge, and will find the proper connection between the crash and the death of the pilot.

```
*(CA '(A SMALL PLANE STUFFED WITH 1500 POUNDS OF MARIJUANA
       CRASHED 10 MILES SOUTH OF HERE AS IT APPROACHED A
       LANDING STRIP KILLING THE PILOT))

(WHEN CONA
     (PROPEL
      ACTOR (PP CLASS (VEHICLE)
                TYPE (AIRPLANE)
                SIZE (SMALL)
                REF (INDEF)
                REL (PTRANS
                     ACTOR (NIL)
                     OBJECT (PP CLASS (PHYSOBJ)
                                TYPE (MJ)
                                AMOUNT
                                (PP CLASS (UNIT)
                                    TYPE (LB)
                                    NUMBER
                                    (NUMBER VAL (*1500*))))
                     TO (INSIDE PART (PREVIOUS))
                     FROM (NIL)
                     TIME (*PAST*)))
      OBJECT (PP CLASS (PHYSOBJ) TYPE (GROUND))
      PLACE (LOC PROX
                (LOC PROX (TOKO))
                DIR (SOUTH)
                DIST
                (PP CLASS (UNIT)
                    TYPE (MILE)
                    NUMBER
                    (NUMBER VAL (*10*))))
       TIME (*PAST*))
      CONB
      (SAPPROACH
       VEH (PP CLASS (NIL) REF (DEF))
       FIELD (LOC LOCTYPE (LANDING-FIELD) REF (INDEF))))

(LEAD-TO
  ANTE (NIL)
  CONSE (HEALTH
         ACTOR
         (PP CLASS (HUMAN) TYPE (PILOT) REF (DEF))
         VAL (-10.)))
```

11.0 CONCLUSION

Our work on conceptual analysis has yielded several different sorts of conclusions. On a theoretical level, our major claim is that

(1) The conceptual analysis process we have presented, as well as previous work, provides evidence for the hypothesis that prior syntactic analysis is unnecessary in language understanding.

Other theoretical claims include:

(2) In order for a language analysis procedure to be able to successfully analyze multi-clause constructions and complex noun groups, and to be capable of handling word-sense ambiguity, it must conform to the criterion of flexibility. This criterion states that an analyzer must be able to operate in both a top-down and bottom-up mode with equal facility.

(3) In order for an analyzer to conform to the criterion of flexibility, it must have a flexible short term memory. We have proposed the use of a simple ordered list of elements, the C-LIST.

(4) In order to conform to the criterion in connection with the problem of word-sense disambiguation, a language analyzer cannot decide among competing expectations and structures solely at the time they are activated. The decision must be made at the time the expectations are considered, and the algorithm must be capable of postponing a decision in some cases.

(5) We have shown that prior syntactic expectations at the start of a sentence are unnecessary if the criterion of flexibility is met.

We have also discussed the following issues:

(6) A partial specification of the notion of syntax from the point of view of its use in a conceptual analyzer (see section 8.1).

(7) The notion of intelligent error correction and its relation to the problem of ambiguity (see section 8.4).

(8) A comparison of some previously described methods for performing word-sense disambiguation, which shows that they are surprisingly similar (see section 9).

In addition, the work described here has contributed to our technical knowledge of conceptual analysis, resulting in improved algorithms. These technical results include:

(9) An improved vocabulary for the tests and actions of requests. (See sections 3 and 6 for an exposition of request tests and actions.) This request vocabulary is in some sense a programming language. Many sections of this paper, particularly section 6, have described the proper use of this language in writing requests.

(10) The use of the recency rule to organize active requests.

(11) How noun-group and clause level analysis can be done using essentially the same control and data structures, if the recency rule is used.

What are the important issues for future work in language analysis? First of all, we must gain a much better understanding of how higher level memory structures can be used in the analysis process. The importance of this question will become apparant, we believe, particularly as we try to address better the problem of word-sense ambiguity. Of course, how well we can apply memory in the parsing process will depend on how well we understand memory. That remains the most difficult, but most crucial, question.

APPENDIX: A Detailed Example

This section will be devoted to discussing a fairly long example in some detail. The example is: "A small plane stuffed with 1500 pounds of marijuana crashed 10 miles south of here." The dictionary entries exhibited are not particularly general, however they illustrate the variety of actions which the analyzer must perform. The example will trace the execution monitor, describing the state of the C-LIST and of the request pools as the sentence is analyzed. We assume that the C-LIST is NIL and there are no request pools at the start.

GET THE NEXT ITEM: In this case the first word of the sentence, "a". The program switches from normal to noun group mode. (The current mode is determined by using Gershman's (1977) heuristics for noun group boundaries.)

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: The entry for "a" looks something like this:

```
(DEF A
 (REQUEST
  [TEST: "find a concept following you"]
  [ACTIONS: "add (REF (INDEF)) to it"]]
```

The request is named REQ1 by the system and activated in POOL-1. Note that this is a request which adds the gap (REF) that it will fill.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ1 is considered but does not fire.

GET THE NEXT ITEM: "small".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: The entry for "small" looks something like this:

```
(DEF SMALL
 (REQUEST
  [TEST: "find PP on C-LIST following you"]
  [ACTIONS: "add (SIZE (SMALL)) to it"]]
```

The request is named REQ2 by the system and activated in POOL-2.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ2 is considered but does not fire.

GET THE NEXT ITEM: "plane".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: The entry for "plane" looks something like this:

```
(DEF PLANE
 (REQUEST
  [TEST: T]
  [ACTIONS:
    "add (PP CLASS (VEHICLE) TYPE (AIRPLANE)) to C-LIST"]]
```

Recall that all structures are added on the end of the C-LIST. The request is named REQ3 by the system and activated in POOL-3.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ3 is considered and fires. The result is that POOL-3 is now empty, and the C-LIST = (CON1) where CON1 = (PP CLASS (VEHICLE) TYPE (AIRPLANE)).

GET THE NEXT ITEM: "stuffed". The program switches from noun group to normal mode. Recall that normal mode requires the execution monitor to immediately consider all active requests. So, the monitor checks all of the request pools, from most recent to least. REQ2 and REQ1 both fire, and the end result is that all the request pools are empty, with CON1 = (PP CLASS (VEHICLE) TYPE (AIRPLANE) SIZE (SMALL) REF (INDEF)).

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: The entry for "stuff" looks something like this:

```
(DEF STUFF
 (REQUEST
  [TEST: T]
  [ACTIONS:

   "STRC := add (PTRANS ACTOR (NIL)
                        OBJECT (NIL)
                        TO (INSIDE PART (NIL)))
            to C-LIST"

   "activate
    (REQUEST
     [TEST: "the next word is WITH"]
     [ACTIONS:
      "activate in the next pool, marked high priority:
       (REQUEST
        [TEST: "find PP on C-LIST and (TO PART) slot of
```

```
            STRO is filled"]
      [ACTIONS: "put it in the OBJECT slot of STRO"])
     (REQUEST
      [TEST: "find PP on C-LIST, preceding STRO"]
      [ACTIONS: "add (REL (STRO)) to it and
                 put it in (TO PART) slot of STRO"]]]
```

The variable STRO (which is replaced by a system-unique symbol) is set to the PTRANS when the request is executed. STRO is used to save the structure for other requests to reference. The request is named REQ4 by the system and activated in POOL-4.

REQUESTS ARE CONSIDERED IN NORMAL MODE: REQ4 is triggered, so now C-LIST = (CON1 CON2), where CON2 is the PTRANS. Another request is activated as REQ5, which is put in the special pool looking for particular words.

GET THE NEXT ITEM: "with".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: REQ5 is triggered. Because of the nature of the action of REQ5, the execution monitor proceeds, noting it must add two new requests, REQ6 and REQ7, to the next request pool.

LOAD THE REQUESTS UNDER THE DICTIONARY ENTRY: In this instance, it doesn't really matter what the requests listed under "with" are, since the requests added by REQ5 are the proper sense of the word in this local context. Hence, when POOL-5 is built, REQ6 and REQ7 are added to it. They have a higher priority than the requests listed under "with" in the dictionary.

CONSIDER REQUESTS IN NORMAL MODE: REQ7, which builds the REL structure, is triggered, and so now C-LIST = (CON1), where CON1 = (PP CLASS (VEHICLE) TYPE (AIRPLANE) SIZE (SMALL) REL (PTRANS ACTOR (NIL) OBJECT (NIL) TO (INSIDE PART (CON1)))). The only remaining active request is REQ6, which strives to fill the OBJECT slot of the PTRANS.

GET THE NEXT ITEM: "1500". The execution monitor changes to noun group mode.

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: The program recognizes that "1500" is a number, and for the sake of convenience generates a corresponding literal atom *1500*, with a definition something like this:

```
(DEF *1500*
 (REQUEST
  [TEST: T]
  [ACTIONs: "add (NUMBER VAL (*1500*) to C-LIST"]]
```

This is named REQ8 by the system, and activated in POOL-6.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ8 is triggered, and the

C-LIST = (CON1 CON3) where CON3 = (NUMBER VAL (*1500*)).

GET THE NEXT ITEM: "pounds".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: the entry for "pound" looks something like this:

```
(DEF POUND
 (REQUEST
  [TEST: T]
  [ACTIONS:

   "STR0 := add (PP CLASS (UNIT)
                    TYPE (LB)
                    NUMBER (NIL))
            to C-LIST"

   "activate
    (REQUEST
     [TEST: "find a number preceding STR0
             on the C-LIST"]
     [ACTIONS: "put it in the NUMBER slot of STR0"]]
    (REQUEST
     [TEST: "next word is OF"]
     [ACTIONS:
      "activate in the next pool, marked high priority:
       (REQUEST
        [TEST: "if you find a PP on C-LIST following STR0"]
        [ACTIONS: "add (AMOUNT STR0) to it"]]]
```

This is named REQ9 and activated in POOL-7 by the system.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ9 is triggered, so that the C-LIST = (CON1 CON3 CON4) where CON4 is the representation for "pounds". Also, the request looking for a number is activated as REQ10 in POOL-8, which should be considered in noun group mode. The other request is activated in the pool for requests testing for particular words as REQ11. REQ10 triggers, and C-LIST = (CON1 CON4), where CON4 = (PP CLASS (UNIT) TYPE (LB) NUMBER (NUMBER VAL (*1500*))).

GET THE NEXT ITEM: "of".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: REQ11 is triggered, and because of the nature of its action, the execution monitor proceeds, noting it must add a new request, REQ12, to the next request pool.

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: Again, since the sense of this word is locally changed, it doesn't matter what they are. However, POOL-9 is built, and REQ12 is activated in it.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ12 is considered, but fails to fire;

GET THE NEXT ITEM: "marijuana".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: the entry for "marijuana" looks something like:

```
(DEF MARIJUANA
 (REQUEST
  [TEST: T]
  [ACTIONs: "add (PP CLASS (PHYSOBJ) TYPE (MJ) to C-LIST"]]
```

This is named REQ13 by the system and activated in POOL-10.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ13 is triggered, so now C-LIST = (CON1 CON4 CON5), where CON5 is the representation for "marijuana".

GET THE NEXT ITEM: "crashed". The execution monitor switches from noun group mode to normal mode, and so immediately considers all active requests. By recency, REQ12 is considered first, and triggers. Now C-LIST = (CON1 CON5) where CON5 = (PP CLASS (PHYSOBJ) TYPE (MJ) AMOUNT (PP CLASS (UNIT) TYPE (LB) NUMBER (NUMBER VAL (*1500*)))); REQ6, which was activated to fill the OBJECT slot of the subordinate PTRANS, is triggered, CON5 fills the OBJECT slot, and C-LIST = (CON1) again. There are no more active requests.

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: The entry for "crash" looks something like this:

```
(DEF CRASH
 (REQUEST
  [TEST: T]
  [ACTIONS:

    STRO := "add (PROPEL ACTOR (NIL)
                         OBJECT (NIL)
                         PLACE (NIL))
             to C-LIST"

    "activate
     (REQUEST
      [TEST: "find a PP which is a physical object
              preceding STRO on the C-LIST"]
      [ACTIONS: "put it in the ACTOR slot of STRO"]
     (REQUEST
      [TEST: "find a PP which is a location on the C-LIST"]
      [ACTIONS: "put it in the PLACE slot of STRO"]]]
```

This is REQ14, activated in POOL-11.

CONSIDER REQUESTS IN NORMAL MODE: REQ14 is triggered, and the C-LIST = (CON1 CON6) where CON6 is the representation for "crash". Also, two new requests, REQ15 and REQ16 are activated in POOL-12. These are considered, and REQ15 is triggered, which places CON1 in the ACTOR slot of CON6, so that C-LIST = (CON6). REQ16 is not triggered.

GET THE NEXT ITEM: "10". Switch to noun group mode.

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: Again, the literal atom *10* is used in place of the number 10. The following entry is constructed:

```
(DEF *10*
 (REQUEST
  [TEST: T]
  [ACTIONS: "build (NUMBER VAL (*10*))"]
```

This is REQ17, activated in POOL-13.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ17 is triggered, so C-LIST = (CON6 CON7) where CON7 = (NUMBER VAL (*10*)).

GET THE NEXT ITEM: "miles".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: In this case:

```
(DEF MILE
 (REQUEST
  [TEST: T]
  [ACTIONS:

   STRO := "add (PP CLASS (UNIT)
                    TYPE (MILE)
                    NUMBER (NIL))
            to C-LIST"

   "activate
    (REQUEST
     [TEST: "find a number preceding STRO on C-LIST"]
     [ACTIONS: "put it in the NUMBER slot of STRO"]]]
```

This is REQ18, activated in POOL-14.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ18 is triggered, so C-LIST = (CON6 CON7 CON8) where CON8 is the representation for "mile". Also, REQ19 is activated in POOL-15, and then also considered. It fires, so that the C-LIST = (CON6 CON8) where CON8 = (PP CLASS (UNIT) TYPE (MILE) NUMBER (NUMBER VAL (*10*))).

GET THE NEXT ITEM: "south".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: The entry for "south" is:

```
(DEF SOUTH
 (REQUEST
  [TEST: T]
  [ACTIONS:

   STRO := "add (PROX PART (NIL)
                     DIR (S)
                     DIST (NIL))
            to C-LIST"

   "activate
    (REQUEST
     [TEST: "find a distance unit preceding
             STRO on the C-LIST"]
     [ACTIONS: "put it in the DIST slot of STRO"]
    (REQUEST
     [TEST: "the next word is OF"]
     [ACTIONS:
      "activate in the next pool, marked as high priority:
       (REQUEST
        [TEST: "find a structure representing a location
                following STRO on the C-LIST"]
        [ACTIONS: "put it in the PART slot of STRO"]]]
```

This is REQ20, activated in POOL-16.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ20 is triggered, and now C-LIST = (CON6 CON8 CON9) where CON9 is the structure representing "south". Also, REQ21 is activated in POOL-17, and REQ22 in the special pool of requests looking for specific words. REQ21 fires, and so now C-LIST = (CON6 CON9) where CON9 = (PROX PART (NIL) DIR (S) DIST (PP CLASS (UNIT) TYPE (MILE) NUMBER (NUMBER VAL (*10*)))).

GET THE NEXT ITEM: "of".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: REQ22 is triggered, but waits until the next request pool is activated to add a request, REQ23.

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: Again, since the meaning of "of" will be locally changed, it doesn't matter what the dictionary request for "of" has in it. However, REQ23 is activated in POOL-18.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ23 is considered but does not fire.

GET THE NEXT ITEM: "here".

CONSIDER REQUESTS LOOKING FOR SPECIFIC WORDS: (There aren't any.)

LOAD THE REQUESTS FROM THE DICTIONARY ENTRY: In this case, the entry looks something like this:

```
(DEF HERE
 (REQUEST
  [TEST: T]
  [ACTIONS: "add (LOC PROX (*HERE*))"]]
```

This becomes REQ24 in POOL-19.

CONSIDER REQUESTS IN NOUN GROUP MODE: REQ24 is triggered, and changes the C-LIST to (CON6 CON9 CON10).

GET THE NEXT ITEM: "period". Switch to normal mode of the execution monitor. This means checking all requests. REQ24 is triggered and C-LIST = (CON6 CON9) where CON9 = (PROX PART (LOC PROX (*HERE*)) DIR (S) DIST (PP CLASS (UNIT) TYPE (MILE) NUMBER (NUMBER VAL (*10*)))). Then REQ17 is considered and triggers, filling the PLACE slot of CON6 with CON9. Since period signals the end of a sentence, there are no more active requests, and the analysis has constructed one complete conceptualization, we are done. C-LIST = (CON6) where

```
CON6 =
(PROPEL ACTOR (CON1)
        OBJECT (NIL)
        PLACE (CON9))

CON1 =
(PP CLASS (VEHICLE)
    TYPE (AIRPLANE) SIZE (SMALL)
    REL (PTRANS ACTOR (NIL)
          OBJECT
           (PP CLASS (PHYSOBJ) TYPE (MJ)
                 AMOUNT
                  (PP CLASS (UNIT)
                        TYPE (LB)
                        NUMBER (NUMBER VAL (*1500*))))
          TO (INSIDE PART (CON1))))

CON9 =
(PROX PART (LOC PROX (*HERE*))
        DIR (S)
        DIST (PP CLASS (UNIT) TYPE (MILE)
                   NUMBER (NUMBER VAL (*10*))))
```

References

Bobrow, D., and Fraser, B. 1969; An Augmented State Transition Network Analysis Procedure, in Proc. IJCAI 1, Washington, D.C., pp. 557-567;

Burton, R. 1976; Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems, BBN Report No. 3453 (ICAI Report No. 3), Bolt Beranek and Newman, Inc., Cambridge, Mass.;

Bruce, B. 1975; Case Systems for Natural Language, in Artificial Intelligence 6, pp. 327-360;

Cullingford, R. 1978; Script Application: Computer Understanding of Newspaper Stories, Research Report No. 116, Dept. of Computer Science, Yale University, New Haven, Conn.;

Davis, R., and King, J. 1975; An Overview of Production Systems, Memo AIM-271, Dept. of Computer Science, Stanford University, Stanford, Ca.;

Gershman, A. 1977; Conceptual Analysis of Noun Groups in English, in Proc. IJCAI 5, Cambridge, Mass., pp. 132-138;

Gershman, A. 1979; Knowledge-Based Parsing, Research Report No. 156, Dept. of Computer Science, Yale University, New Haven, Conn.;

Hayes, P. 1977; Some Association-Based Techniques for Lexical Disambiguation by Machine, Technical Report No. 25, Dept. of Computer Science, University of Rochester, Rochester, N.Y.;

Hoogenraad, R., Grieve, R., Baldwin, P., and Campbell, R. 1978; Comprehension as an Interactive Process, in R. Campbell and P. Smith, eds., Recent Advances in the Psychology of Language: Language Development and Mother-Child Interaction, Plenum Press, N.Y., pp. 163-186;

Kaplan, R. 1975; On Process Models for Sentence Analysis, in D. Norman and D. Rumelhart, eds., Explorations in Cognition, W. H. Freeman Comp., San Francisco, Ca., pp. 117-135;

Katz, J., and Fodor, J. 1963; The Structure of a Semantic Theory, in Language 39, pp. 170-210;

Kolodner, J. 1978; Memory Organization for Natural Language Data-Base Inquiry, Research Report No. 142, Dept. of Computer Science, Yale University, New Haven, Conn.;

Lehnert, W., and Burstein, M. 1979; The Role of Object Primitives in Natural Language Processing, Research Report No. 162, Dept. of Computer Science, Yale University, New Haven, Conn.;

Marcus, M. 1975; Diagnosis as a Notion of Grammar, in Proc. TINLAP 1, Cambridge, Mass., pp. 6-10;

Marcus, M. 1979; A Theory of Syntactic Recognition for Natural Language, in P. Winston and R. Brown, eds., Artificial Intelligence: An MIT Perspective, The MIT Press, Cambridge, Mass., pp.191-229;

Newell, A. 1973; Production Systems: Models of Control Structures, in W. Chase, ed., Visual Information Processing, Academic Press, New York, pp. 463-526;

Riesbeck, C. 1975; Conceptual Analysis, in R. Schank, ed., Conceptual Information Processing, North-Holland Publishing Comp., Amsterdam, pp. 83-156;

Riesbeck, C., and Schank, R. 1976; Comprehension by Computer: Expectation-Based Analysis of Sentences in Context, Research Report No. 78, Dept. of Computer Science, Yale University, New Haven, Conn.;

Schank, R., Tesler, L., and Weber, S. 1970; Spinoza II: Conceptual Case-Based Natural Language Analysis, Memo AIM-109, Dept. of Computer Science, Stanford University, Stanford, Ca.;

Schank, R., ed. 1975; Conceptual Information Processing, North-Holland Publishing Comp., Amsterdam;

Schank, R., and Abelson, R. 1977; Scripts, Plans, Goals, and Understanding, Lawrence Erlbaum Associates, Hillsdale, N.J.;

Schank, R., and Selfridge, M. 1977; How to Learn/What to Learn, in Proc. IJCAI 5, Cambridge, Mass., pp. 8-14;

Selfridge, M. 1979; A Process Model of Language Acquisition, forthcoming Ph.D. thesis, Dept. of Computer Science, Yale University, New Haven, Conn.;

Small, S. 1978; Conceptual Language Analysis for Story Comprehension, Technical Report No. 663, Dept. of Computer Science, Univ. of Maryland, College Park, Md.;

Thorne, J., Bratley, P., and Dewar H. 1968; The Syntactic Analysis of English by Machine, in D. Michie, ed., Machine Intelligence 3, American Elsevier Publishing Comp., New York, pp. 281-309;

Wilks, Y. 1973; An Artificial Intelligence Approach to Machine Translation, in R. Schank and K. Colby, eds., Computer Models of Thought and Language, W. H. Freeman and Co., San Francisco, pp. 114-151;

Wilks, Y. 1975; Seven Theses on Artificial Intelligence and Natural Language, Research Report No. 17, Instituto per gli Studi Semantici e Cognitivi, Castagnola, Switzerland;

Wilks, Y. 1976; Parsing English I and II, in E. Charniak and Y. Wilks, eds., Computational Semantics, North-Holland Publishing Co., Amsterdam, pp. 89-100 and 155-184;

Winograd, T. 1972; Understanding Natural Language, Academic Press, New York;

Woods, W. 1970; Transition Network Grammars for Natural Language Analysis, in Comm. ACM 13, pp. 591-606;